The Evolution of In-Context Reasoning: From Chain-of-Thought to Iterative Self-Improvement

Yuxuan Long, Longhan Lin & Taumas Yang

Content

1. **Discovery:** We can unlock reasoning by prompting models to “show their work” (**Chain-of-Thought**).
2. **Robustness:** We can make this reasoning more reliable by exploring multiple reasoning paths and finding a consensus (**Self-Consistency**).
3. **Agency:** We can go a step further and have the model critique and improve its own work in a loop (**Self-Refine**).
4. **Meta-Analysis:** All these methods fall under the umbrella of “test-time compute.” How does spending more compute at inference compare to simply training a bigger model? (**Scaling LLM Test-Time Compute**).

Content

1. **Discovery:** We can unlock reasoning by prompting models to “show their work” (**Chain-of-Thought**).
2. **Robustness:** We can make this reasoning more reliable by exploring multiple reasoning paths and finding a consensus (**Self-Consistency**).
3. **Agency:** We can go a step further and have the model critique and improve its own work in a loop (**Self-Refine**).
4. **Meta-Analysis:** All these methods fall under the umbrella of “test-time compute.” How does spending more compute at inference compare to simply training a bigger model? (**Scaling LLM Test-Time Compute**).

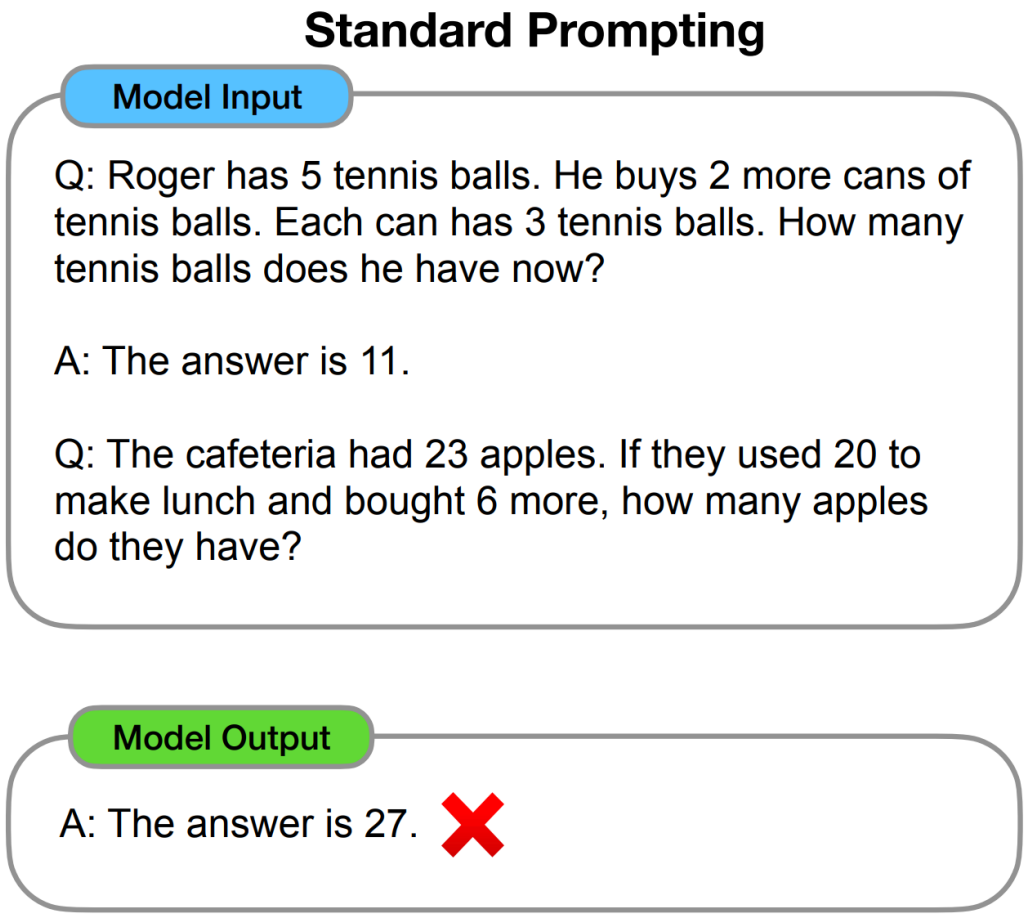
Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

Jason Wei et al. (2022)

The Challenge: Complex Reasoning in a Single Step

Why did it fail?

- The model likely latched onto the numbers (23, 20, 6) and performed a plausible but incorrect operation (e.g., 23 - 20 = 3; 23 + (6-something) ≈ 27? Or just 20+6+initial confusion ≈ 27).
- It tried to solve a multi-step problem (subtract, then add) in a single, intuitive leap.
- This approach is not robust and has a flat scaling curve–making the model bigger doesn't reliably fix this kind of error.

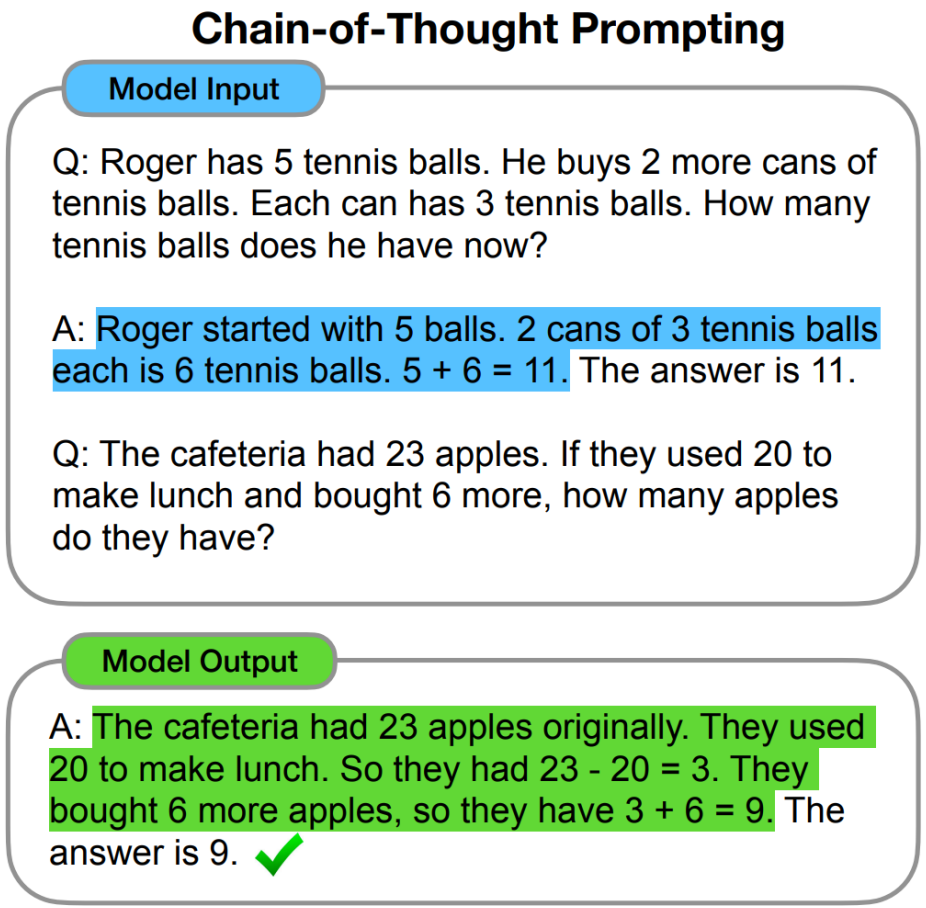


Prior Solutions and Limitations

- Method 1: Finetuning with Rationales
  - Train a model on a large dataset of problems and human-written explanations.
  - Limitation: Requires creating massive, expensive, and high-quality datasets.
- Method 2: Standard Few-Shot Prompting
  - Give a few questions → Answer examples in the prompt.
  - Limitation: As we saw, this doesn’t work well for reasoning tasks.

The Solution: Chain-of-Thought Prompting

- It's a **few-shot** prompting technique where the exemplars given to the model don't just show the final answer but also include the **intermediate reasoning steps** used to get there.
- Show the **process**.



Why Might CoT Work?

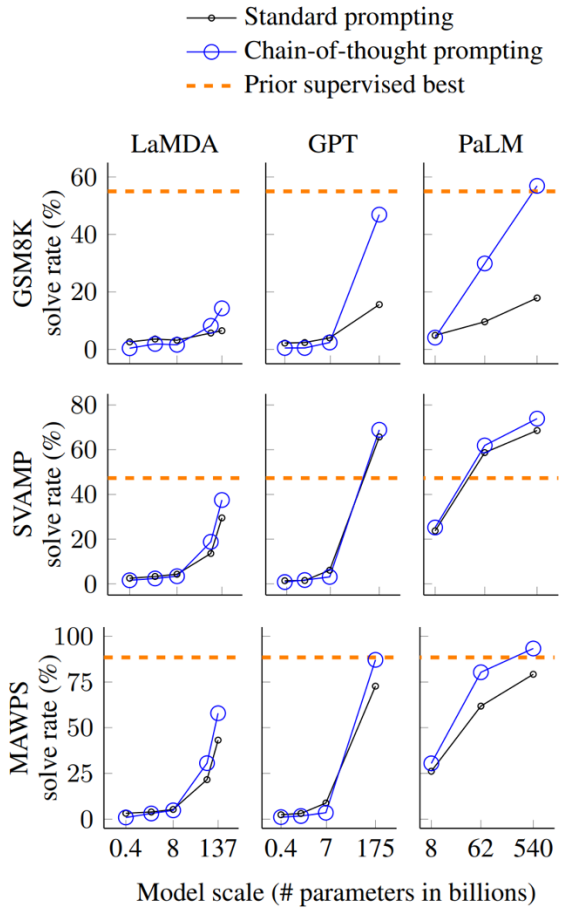
- **Decomposition:** Allows the model to break a multi-step problem into smaller, manageable steps.
- **Allocation of Computation:** The model can spend more “thought” (i.e., generate more tokens) on more complex problems.
- **Interpretability:** Provides a window into the model’s “thought process,” allowing for debugging.
- **Generality:** Applicable to any task that can be solved via language.

Experimental Setup

- **Arithmetic:** GSM8K, SVAMP, etc. (Math problems)
- **Commonsense:** StrategyQA, CSQA (Everyday logic)
- **Symbolic:** Last Letter Concatenation, Coin Flip (Abstract manipulation)
- **Models:** GPT-3, LaMDA, PaLM, UL2 20B, Codex
- **Method:** Few-shot prompting (e.g., 8 exemplars)

Key Result #1: Gains in Arithmetic Reasoning

- The huge performance gap between standard prompting (black) and CoT prompting (blue) for large models.
- For PaLM 540B on GSM8K, performance jumps from 17.9% to 56.9%.
- A CoT-prompted model can outperform a specially finetuned model.



Key Result #2: Reasoning is an Emergent Ability of Scale

- For smaller models (under ~100B parameters), CoT provides no benefit.

Question

Tracy used a piece of wire 4 feet long to support tomato plants in the garden. The wire was cut into pieces 6 inches long. How many pieces did she obtain?

62B Model Output

She cut the wire into 6 inch pieces. This means she got 4 * 6 = 24 pieces. The answer is 24. **(Incorrect: semantic understanding error)**

540B Model Output

The wire was 4 feet long. This means it was 4 * 12 = 48 inches long. It was cut into pieces 6 inches long. This means she obtained 48 / 6 = 8 pieces. The answer is 8. **(Correct)**

Question

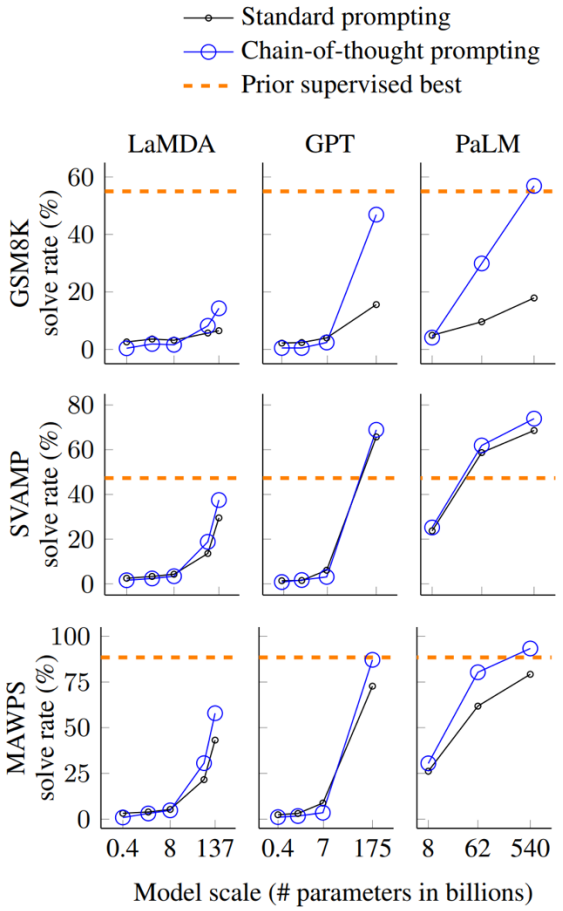
Tom's ship can travel at 10 miles per hour. He is sailing from 1 to 4 PM. He then travels back at a rate of 6 mph. How long does it take him to get back?

62B Model Output

He travels from 1 to 4 PM. That means he travels for 3 hours. He travels back at 6 mph. That means he travels for 3 * 6 = 18 miles. The answer is 18. **(Incorrect: semantic understanding error)**

540B Model Output

He travels at 10 mph from 1 to 4 PM. This means he travels 3 hours. 3 hours at 10 mph means he travels 3 * 10 = 30 miles. He then travels back at 6 mph. This means he travels 6 miles per hour. He has to travel 30 miles, so it takes him 30 / 6 = 5 hours. The answer is 5. **(Correct)**



Ablation Studies

- **Equation only:** Just providing the math equation isn't enough. The natural language steps are key.
- **Variable compute only (...):** It‘s not just about letting the model “think longer.” The content of the thoughts matters.
- **Reasoning after answer:** The reasoning must happen before the answer to guide the model.

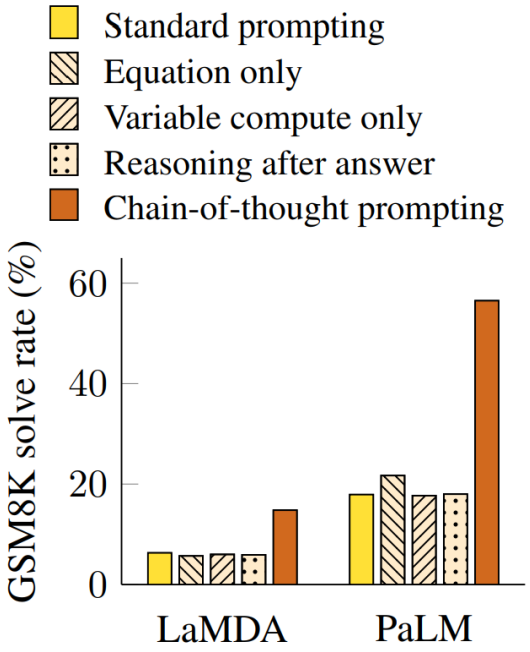


Figure 5: Ablation study for different variations of prompting using LaMDA 137B and PaLM 540B. Results for other datasets are given in Appendix Table 6 and Table 7.

Robustness Analysis

- **Question:** Is CoT sensitive to the prompt?
- **Different Annotators:** Works even when different people write the reasoning steps.
- **Different Exemplars:** Works with different examples, even from other datasets.
- **Different Order:** Relatively robust to the order of examples in the prompt.

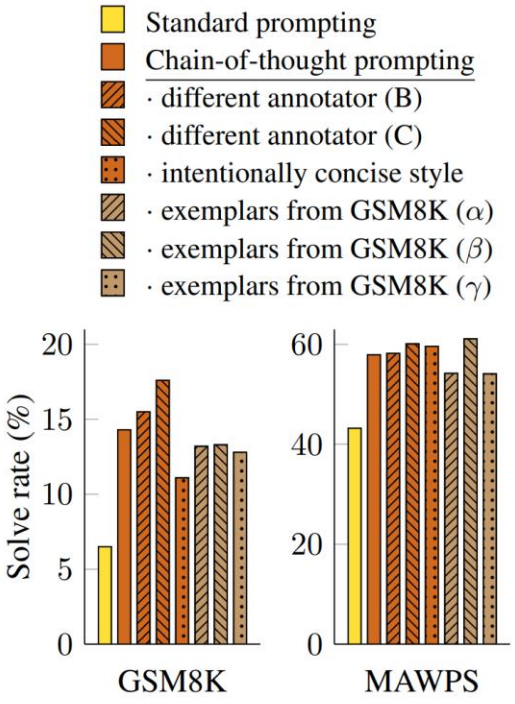
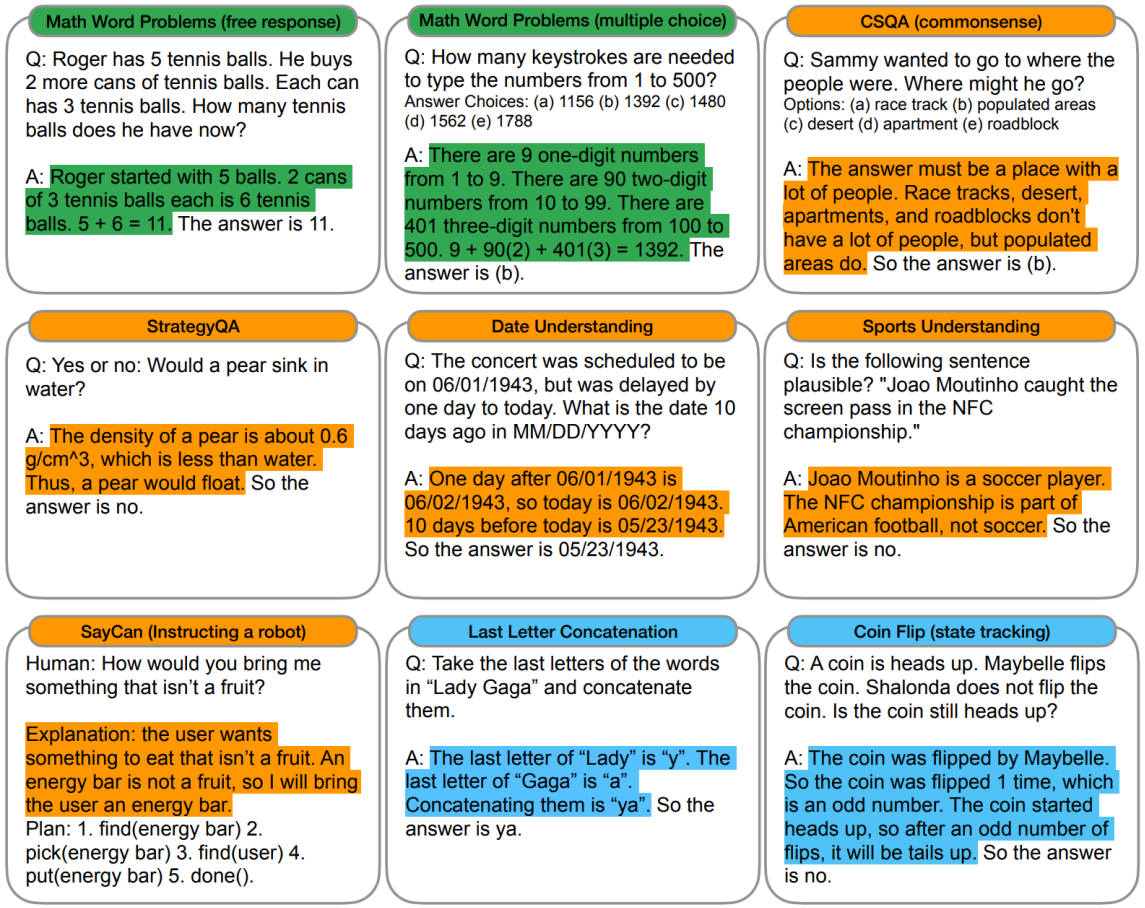
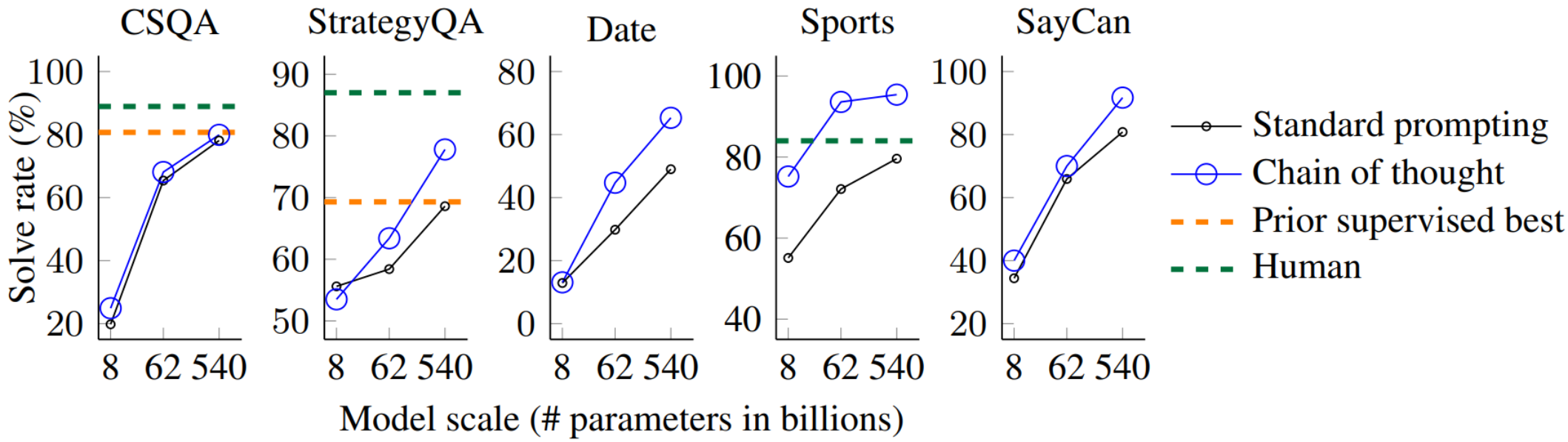


Figure 6: Chain-of-thought prompting has variance for different prompt examples (as expected) but outperforms standard prompting for various annotators as well as for different exemplars.

Generalizing to Other Tasks



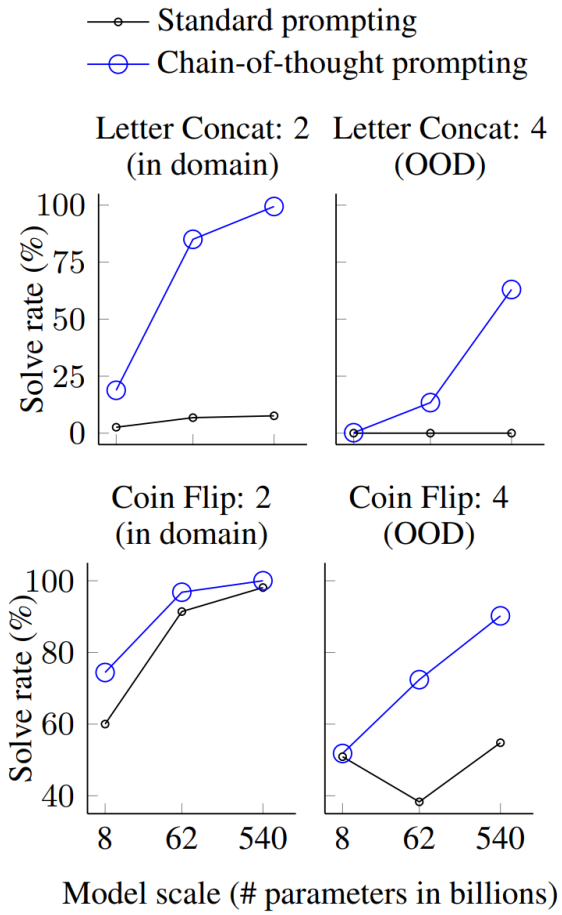
Generalizing to Other Tasks



- Increased commonsense reasoning abilities

Generalizing to Other Tasks

- Increased symbolic reasoning abilities



Why will smaller models not work?

- Generation of Illogical Chains of Thought
- Failure on Simple Symbolic Tasks
- Inherently Weaker Arithmetic Abilities
- Inability to Produce Parsable Answers

- “In summary, the success of chain-of-thought reasoning as a result of model scale is a complicated phenomenon that likely involves a variety of emergent abilities (semantic understanding, symbol mapping, staying on topic, arithmetic ability, faithfulness, etc).”

Limitations and Future Work

- Fallible Reasoning: The generated chain of thought is not guaranteed to be factually or logically correct.
- Scale Requirement: This powerful technique is currently only accessible for massive, computationally expensive models.
- New Lower Bound: Standard prompting should be seen as a lower bound on LLM capabilities. How we interact with models can unlock latent abilities.
- Zero-shot CoT (e.g., just adding “Let’s think step by step”).
- Automating CoT generation.
- Inducing reasoning in smaller, more efficient models.

Conclusion

1. Chain-of-Thought prompting is a simple method to unlock complex reasoning in LLMs.
2. It works by showing the model how to reason step-by-step in few-shot examples.
3. This ability is an emergent property of model scale.
4. CoT establishes new state-of-the-art results on reasoning benchmarks, even outperforming finetuned models.

Beyond Conclusion

- Chain-of-Thought doesn’t prove that LLMs “reason” in the conscious, abstract, human sense.
- It demonstrates that they can move beyond simple input-output pattern matching to imitate a reasoning process.
- This procedural imitation is so effective that it becomes a powerful tool for solving complex problems.
- Constraining the solution space increases the chance of a correct answer.

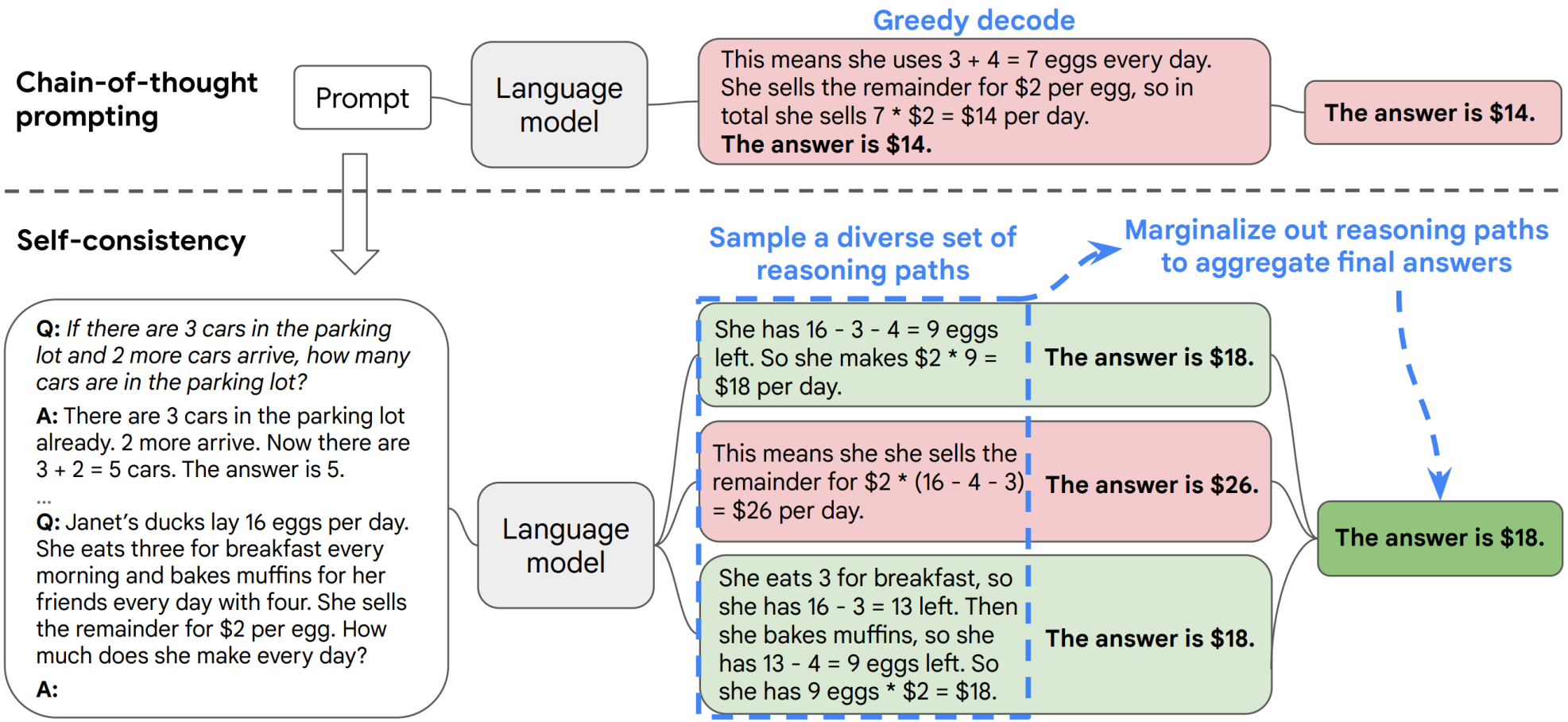
Content

1. **Discovery:** We can unlock reasoning by prompting models to “show their work” (**Chain-of-Thought**).
2. **Robustness:** We can make this reasoning more reliable by exploring multiple reasoning paths and finding a consensus (**Self-Consistency**).
3. **Agency:** We can go a step further and have the model critique and improve its own work in a loop (**Self-Refine**).
4. **Meta-Analysis:** All these methods fall under the umbrella of “test-time compute.” How does spending more compute at inference compare to simply training a bigger model? (**Scaling LLM Test-Time Compute**).

Self-Consistency Improves Chain of Thought Reasoning in Language Models

Xuezhi Wang et al. (2023)

Self-consistency method



Question

- They explore multiple reasoning paths to select the most common answer and improve QA accuracy. They mention that this results in a large computational cost as most of the benefit comes from 5-10 replicates. Could a more restrictive beam search approach be used here to improve compute cost?
- NO, the way to generate is multiple answers, no matter how these answer are similar, more distinct, more accurate.
- How can it be ensured that the different reasoning paths identified by self-consistency represent diverse methodologies, as opposed to various restatements of a uniform reasoning method?
- This is Generative large models, Logical large model, we don’t know.

The self-consistency method three steps

- Prompt a language model using chain-of-thought (CoT) prompting
- Replace the “greedy decode” in CoT prompting by sampling from the language model’s decoder to generate a diverse set of reasoning paths
- Marginalize out the reasoning paths and aggregate by choosing the most consistent answer in the final answer set

Question

- Could only reasoning paths sufficiently dissimilar from the other paths be included in the final aggregation step?
- No, all the result will be sum, no matter how they same both result or token set.

Different answer aggregation strategies

	GSM8K	MultiArith	AQuA	SVAMP	CSQA	ARC-c
Greedy decode	56.5	94.7	35.8	79.0	79.0	85.2
Weighted avg (unnormalized)	56.3 ± 0.0	90.5 ± 0.0	35.8 ± 0.0	73.0 ± 0.0	74.8 ± 0.0	82.3 ± 0.0
Weighted avg (normalized)	22.1 ± 0.0	59.7 ± 0.0	15.7 ± 0.0	40.5 ± 0.0	52.1 ± 0.0	51.7 ± 0.0
Weighted sum (unnormalized)	59.9 ± 0.0	92.2 ± 0.0	38.2 ± 0.0	76.2 ± 0.0	76.2 ± 0.0	83.5 ± 0.0
Weighted sum (normalized)	74.1 ± 0.0	99.3 ± 0.0	48.0 ± 0.0	86.8 ± 0.0	80.7 ± 0.0	88.7 ± 0.0
Unweighted sum (majority vote)	74.4 ± 0.1	99.3 ± 0.0	48.3 ± 0.5	86.6 ± 0.1	80.7 ± 0.1	88.7 ± 0.1

Question

- Self-consistency improves LLMs' reasoning accuracy be sampling multiple reasoning paths and aggerating the answers. And in the paper, they show that this strategy can improve the performance on multiple benchmarks. The first thing that I have not fully captured is, what does "weighted average" in Section 2 mean?
- See, next. weights come from the conditional probability of each generated output

Normalized weighted processing

$$p(\boldsymbol{r}_i, \boldsymbol{a}_i|\text{prompt}, \text{question}) = \exp\left(\frac{1}{K}\sum_{k=1}^{K}\log P(t_k|\text{prompt}, \text{question}, t_1, \ldots, t_{k-1})\right)$$

- $\boldsymbol{r}_i$ An additional latent variable
- $\boldsymbol{a}_i$ The generated answers
- $\log p()$ The log probability
- K Total number of tokens
- k k-th token

Question:

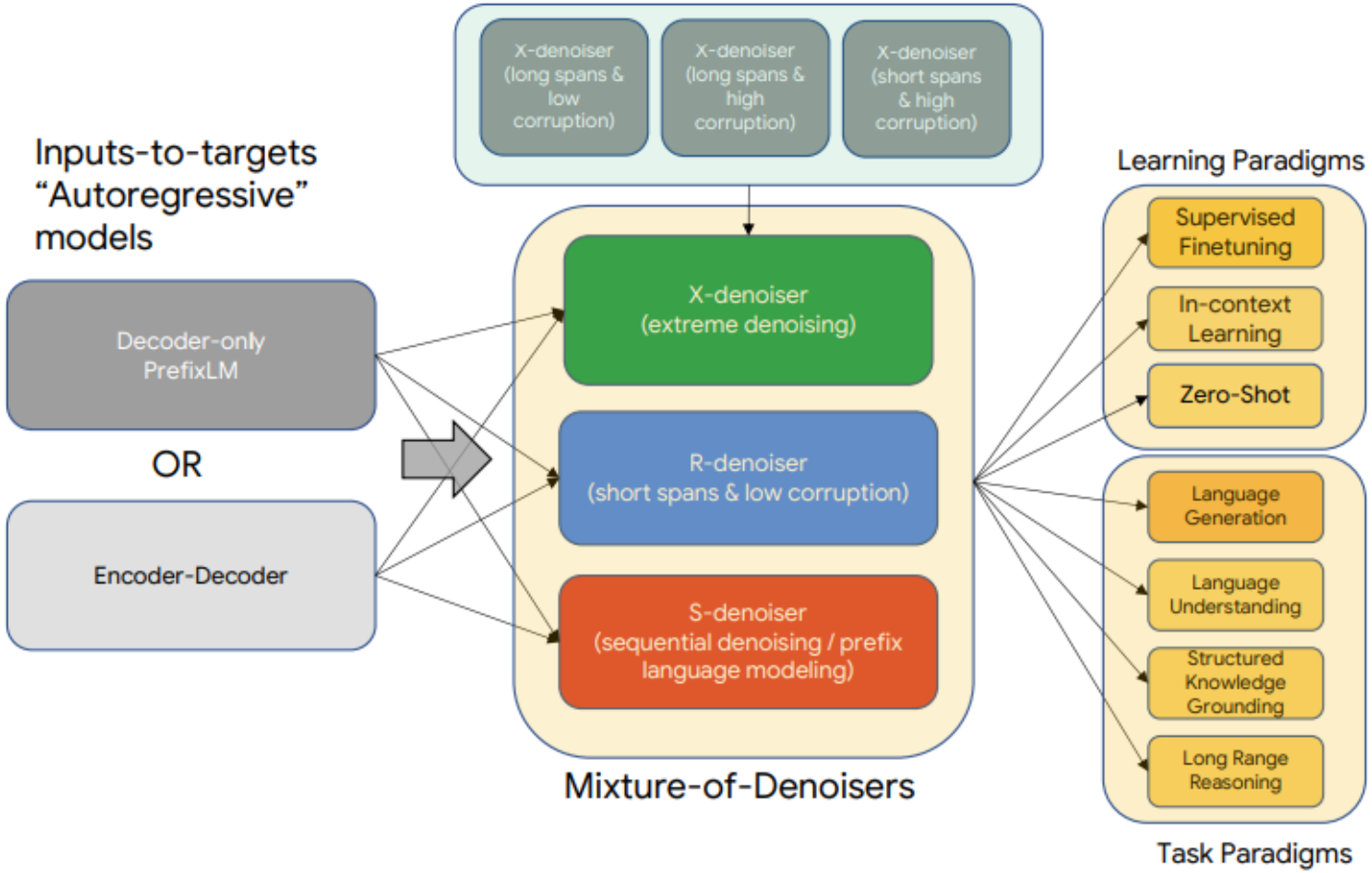
- For self-consistency, why the conditional probability of the response is a suitable way to evaluate?
- response (i.e. series of tokens) is actually a conditional probability chain （How to generate a large language model）
- quality of the response = the product of the conditional probabilities of each token, larger, more "trusted“
- Underflow, solve Log Addition; long sentence log smaller, solve Normalization; last solve log, use exp.

Tasks and datasets

- Arithmetic reasoning:
  - Math Word Problem Repository(AddSub, MultiArith, ASDiv)
  - AQUA-RAT (GSM8K, SVAMP)
- Commonsense reasoning:
  - CommonsenseQA
  - StrategyQA
  - AI2 Reasoning Challenge (ARC)
- Symbolic Reasoning:
  - Last letter concatenation
  - Coinflip

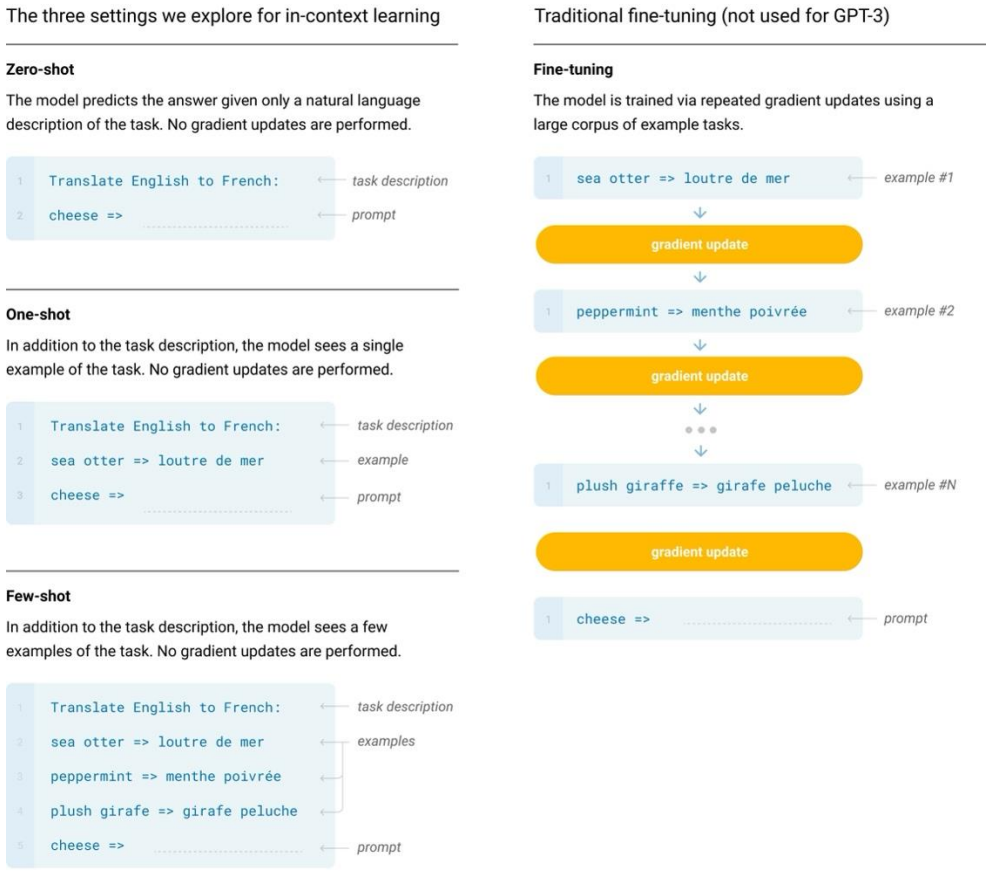
Language models and prompts

- UL2
  - 20-billion parameters
  - ≥ GPT3 on zero-shot
  - compute-friendly



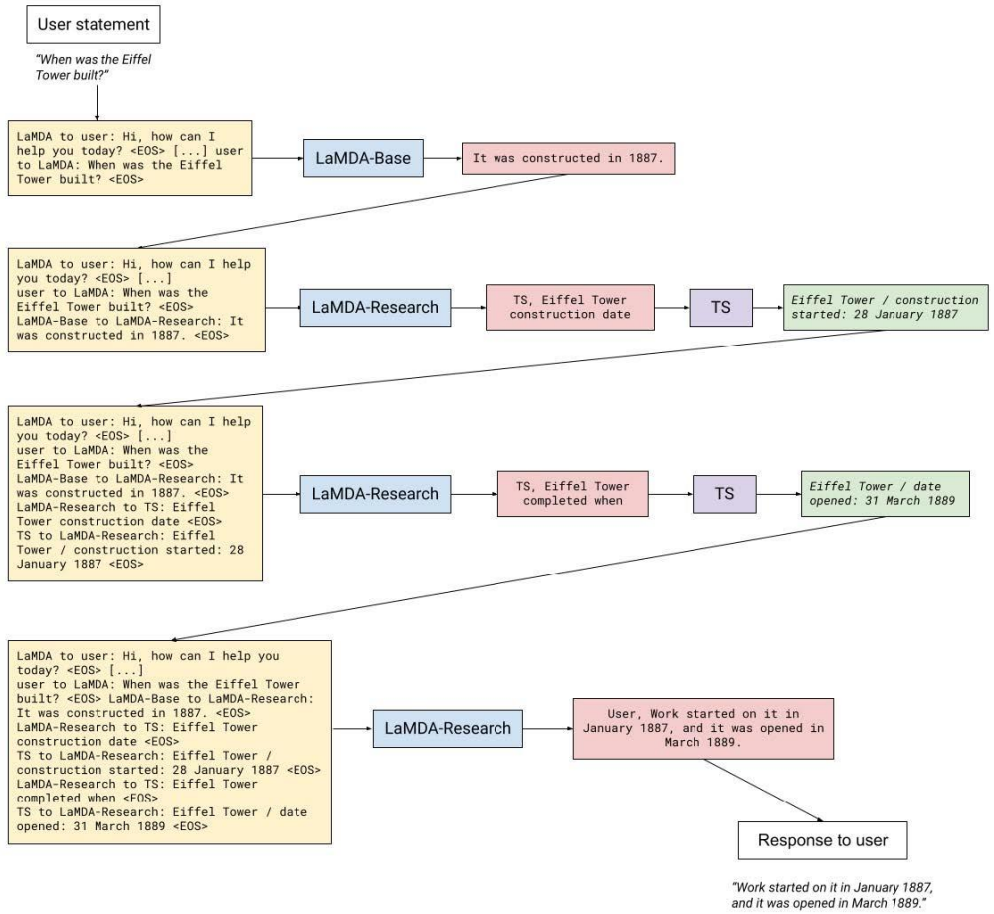
Language models and prompts

- GPT-3
  - 175-billion parameters
  - two public engines
    - code-davinci-001
    - code-davinci-002



Language models and prompts

- LaMDA-137B
  - 137-billion parameters
  - dense left-to-right
  - decoder-only
  - pre-trained web documents



Language models and prompts

- PaLM-540B
  - 540-billion parameters
  - dense left-to-right
  - decoder-only
  - pre-trained 780 billion tokens

Model Summary	
Model Architecture	Dense decoder-only model with 540 billion parameters. Transformer model architecture with variants to speed up training and inference. For details, see Model Architecture (Section 2).
Input(s)	The model takes text as input.
Output(s)	The model generates text as output.
Usage	
Application	The primary use is research on language models, including: research on NLP applications like machine translation and question answering, advancing fairness and safety research, and understanding limitations of current LLMs. Within Google, PaLM is being used for research on a variety of open-ended text and code generation tasks, including reasoning (Section 6.3) and code synthesis and understanding (Section 6.4).
Known Caveats	Gopher (Rae et al., 2021) describes safety benefits and safety risks associated with large language models, including PaLM. These risks include uses of language models for language generation in harmful or deceitful settings. PaLM should not be used for downstream applications without a prior assessment and mitigation of the safety and fairness concerns specific to the downstream application. In particular, we recommend focusing mitigation efforts at the downstream application level rather than at the pretrained level.
System Type	
System Description	This is a standalone model.
Upstream Dependencies	None.
Downstream Dependencies	None.
Implementation Frameworks	
Hardware & Software: Training	Hardware: TPU v4 (Jouppi et al., 2020). Software: T5X (t5x, 2021), JAX (Bradbury et al., 2018), Pathways (Barham et al., 2022). For details, see Training Infrastructure (Section 4).
Hardware & Software: Deployment	Hardware: TPU v4 (Jouppi et al., 2020). Software: T5X (t5x, 2021).
Compute Requirements	Reported in Compute Usage (Section B).
Model Characteristics	
Model Initialization	The model is trained from a random initialization.
Model Status	This is a static model trained on an offline dataset.
Model Stats	The largest PaLM model has 540 billion dense parameters. We have also trained 8 billion and 62 billion parameter models.
Data Overview	
Training Dataset	See Datasheet (Appendix D) for the description of datasets used to train PaLM.
Evaluation Dataset	We evaluate the PaLM family of models on a wide variety of tasks. Specifically, we evaluate the models on English Natural Language Processing (NLP) tasks (Section 6.1), tasks from BIG-bench (BIG-bench collaboration, 2021), reasoning tasks (Section 6.3), code completion tasks (Section 6.4), multilingual generation and question answering tasks (Section 6.6), translation tasks (Section 6.5), and bias and toxicity benchmarks (Rudinger et al., 2018; Gehman et al., 2020).
Fine-tuning Dataset	We include finetuning results on SuperGLUE (?), tasks from GEM (Gehrmann et al., 2021), and TyDiQA (Clark et al., 2020). We also finetune on a code dataset and share results on the finetuned model on code synthesis tasks.
Evaluation Results	
Benchmark Information	• Fewshot: English Natural Language Processing (NLP) tasks (Section 6.1), BIG-bench (Section 6.2), Reasoning (Section 6.3), Code (Section 6.4), GEM (Section 6.6), Translation (Section 6.5), Multi-lingual Question Answering (Section 6.7) • Finetuning: SuperGLUE (Section 6.1.2), GEM (Section 6.6), TyDiQA (Section 6.7). • Responsible AI: Co-occurrence, Winogender (Section 10.1.1), RealToxicity (Section 10.2). • Data contamination (Section 8)
Evaluation Results	Reported in Evaluation (Section 6).
Model Usage & Limitations	
Sensitive Use	PaLM is capable of open-ended text generation. This model should not be used for any of the unacceptable language model use cases, e.g., generation of toxic speech.
Known Limitations	PaLM is designed for research. The model has not been tested in settings outside of research that can affect performance, and it should not be used for downstream applications without further analysis on factors in the proposed downstream application.
Ethical Considerations & Risks	Reported in Ethical Considerations (Section 11).

Sampling scheme

- UL2-20B & LaMDA-137B
  - temperature sampling
  - T = 0.5
  - Top k (k = 40)
- PaLM-540B
  - T = 0.7
  - k = 40
- GPT-3
  - T = 0.3
  - without k

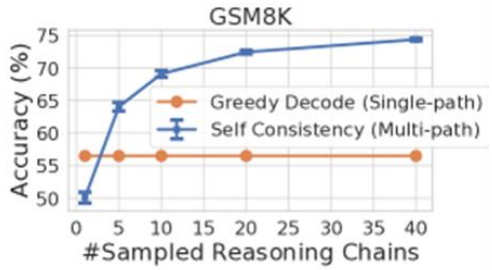
Arithmetic reasoning accuracy

	Method	AddSub	MultiArith	ASDiv	AQuA	SVAMP	GSM8K
	Previous SoTA	94.9[a]	60.5[a]	75.3[b]	37.9[c]	57.4[d]	35[e] / 55[g]
UL2-20B	CoT-prompting	18.2	10.7	16.9	23.6	12.6	4.1
	Self-consistency	24.8 (+6.6)	15.0 (+4.3)	21.5 (+4.6)	26.9 (+3.3)	19.4 (+6.8)	7.3 (+3.2)
LaMDA-137B	CoT-prompting	52.9	51.8	49.0	17.7	38.9	17.1
	Self-consistency	63.5 (+10.6)	75.7 (+23.9)	58.2 (+9.2)	26.8 (+9.1)	53.3 (+14.4)	27.7 (+10.6)
PaLM-540B	CoT-prompting	91.9	94.7	74.0	35.8	79.0	56.5
	Self-consistency	93.7 (+1.8)	99.3 (+4.6)	81.9 (+7.9)	48.3 (+12.5)	86.6 (+7.6)	74.4 (+17.9)
GPT-3Code-davinci-001	CoT-prompting	57.2	59.5	52.7	18.9	39.8	14.6
	Self-consistency	67.8 (+10.6)	82.7 (+23.2)	61.9 (+9.2)	25.6 (+6.7)	54.5 (+14.7)	23.4 (+8.8)
GPT-3Code-davinci-002	CoT-prompting	89.4	96.2	80.1	39.8	75.8	60.1
	Self-consistency	91.6 (+2.2)	100.0 (+3.8)	87.8 (+7.6)	52.0 (+12.2)	86.8 (+11.0)	78.0 (+17.9)

Question

- Under what conditions is self-consistent majority voting most effective?
- ASDiv question, GPT-3Code-davinci-002 : 100 Accuracy

Question



- In the paper, the authors show that self-consistency significantly improves performance on math word problems like GSM8K. Why is this type of task particularly suitable for demonstrating the benefits of self-consistency?
- Author :In Table 9, we further show that self-consistency is quite robust to different sets of input prompts. We manually wrote 3 different sets of chain-of-thought as prompts to the model. Across all sets of prompts, self-consistency yields consistent gains over the original CoT approach.
- Robust, More redundant information, more tokens, see the Temperature sampling.

	Prompt set 1 (used in the main text)	Prompt set 2	Prompt set 3
CoT (Wei et al., 2022)	56.5	54.6	54.0
Self-consistency	74.4 (+17.9)	72.1 (+17.5)	70.4 (+16.4)

Arithmetic reasoning accuracy

	Method	CSQA	StrategyQA	ARC-e	ARC-c	Letter (4)	Coinflip (4)
	Previous SoTA	91.2^a	73.6^b	86.4^c	75.0^c	N/A	N/A
UL2-20B	CoT-prompting	51.4	53.3	61.6	42.9	0.0	50.4
	Self-consistency	55.7 (+4.3)	54.9 (+1.6)	69.8 (+8.2)	49.5 (+6.8)	0.0 (+0.0)	50.5 (+0.1)
LaMDA -137B	CoT-prompting	57.9	65.4	75.3	55.1	8.2	72.4
	Self-consistency	63.1 (+5.2)	67.8 (+2.4)	79.3 (+4.0)	59.8 (+4.7)	8.2 (+0.0)	73.5 (+1.1)
PaLM -540B	CoT-prompting	79.0	75.3	95.3	85.2	65.8	88.2
	Self-consistency	80.7 (+1.7)	**81.6 (+6.3)**	**96.4 (+1.1)**	**88.7 (+3.5)**	70.8 (+5.0)	91.2 (+3.0)
GPT-3 Code-davinci -001	CoT-prompting	46.6	56.7	63.1	43.1	7.8	71.4
	Self-consistency	54.9 (+8.3)	61.7 (+5.0)	72.1 (+9.0)	53.7 (+10.6)	10.0 (+2.2)	75.9 (+4.5)
GPT-3 Code-davinci -002	CoT-prompting	79.0	73.4	94.0	83.6	70.4	99.0
	Self-consistency	81.5 (+2.5)	79.8 (+6.4)	96.0 (+2.0)	87.5 (+3.9)	**73.4 (+3.0)**	**99.5 (+0.5)**

Question

- Another thing I am interested in is, what is the relationship between the base model's capability and the improvement that Self-consistency can bring? If the base model is weak or a problem is too hard, can self-consistency actually improve the performance?
- No, although the base performance not well, self-consistency could still have improve.

Self-Consistency Helps Chain-of-Thought

	ANLI R1 / R2 / R3	e-SNLI	RTE	BoolQ	HotpotQA (EM/F1)
Standard-prompting (no-rationale)	69.1 / 55.8 / 55.8	85.8	84.8	71.3	27.1 / 36.8
CoT-prompting (Wei et al., 2022)	68.8 / 58.9 / 60.6	81.0	79.1	74.2	28.9 / 39.8
Self-consistency	**78.5 / 64.5 / 63.4**	**88.4**	**86.3**	**78.4**	**33.8 / 44.6**

Cause: CoT forces a “chain of reasoning”
Reason: introduce errors or noise
Result: worse than a direct answer

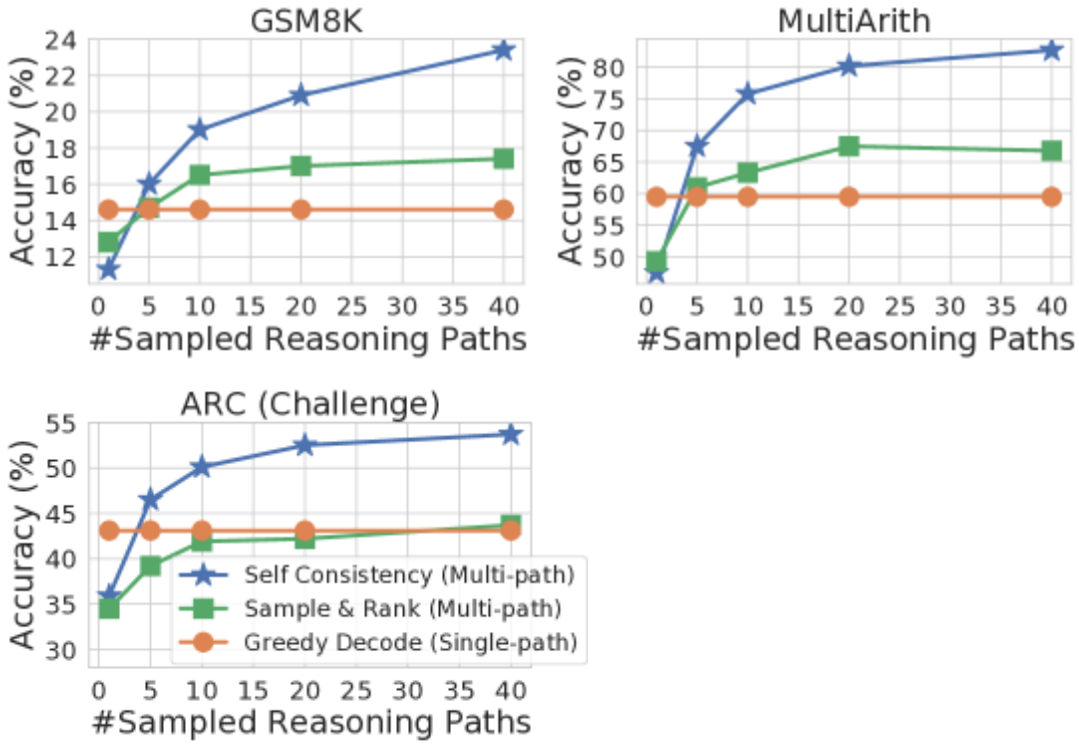
Self-consistency VS Sample-and-Rank

GPT-3 code-davinci-001

1. Multiple sequences are sampled from the decoder
2. Ranked according to each sequence's log probability

Result: As the number of samples increases, Sample-and-rank is indeed better than greedy

The improvement is much smaller than Self-consistency



Self-consistency VS Beam Search

	Beam size / Self-consistency paths	1	5	10	20	40
AQuA	Beam search decoding (top beam)	23.6	19.3	16.1	15.0	10.2
	Self-consistency using beam search	23.6	19.8 ± 0.3	21.2 ± 0.7	24.6 ± 0.4	24.2 ± 0.5
	Self-consistency using sampling	19.7 ± 2.5	**24.9 ± 2.6**	**25.3 ± 1.8**	**26.7 ± 1.0**	**26.9 ± 0.5**
MultiArith	Beam search decoding (top beam)	10.7	12.0	11.3	11.0	10.5
	Self-consistency using beam search	10.7	11.8 ± 0.0	11.4 ± 0.1	12.3 ± 0.1	10.8 ± 0.1
	Self-consistency using sampling	9.5 ± 1.2	11.3 ± 1.2	**12.3 ± 0.8**	**13.7 ± 0.9**	**14.7 ± 0.3**

Self-consistency VS Beam Search

- The number of beams increases accuracy, even decreases
  - Beam search tends to be deterministic
  - Generated paths lack diversity
- Beam search as a sampling method
  - Result better than only beam search
- Self-consistency using sampling still the best
- Diversity is key. Beam search lacks diversity
  - Limits SC effectiveness
- Random sampling leads to more dispersed paths, which makes voting more reliable.

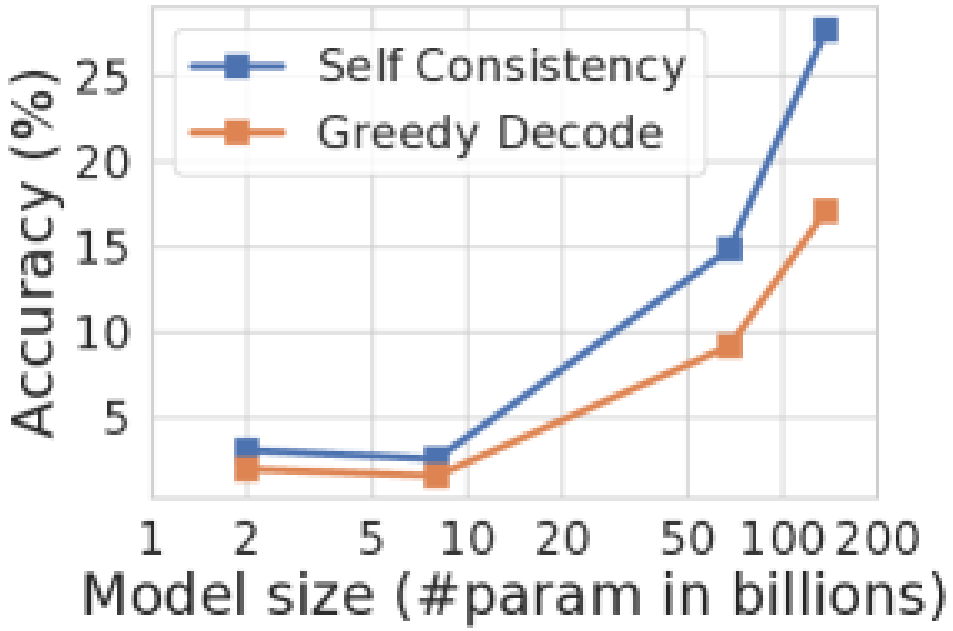
Self-consistency VS Sample-and-Rank

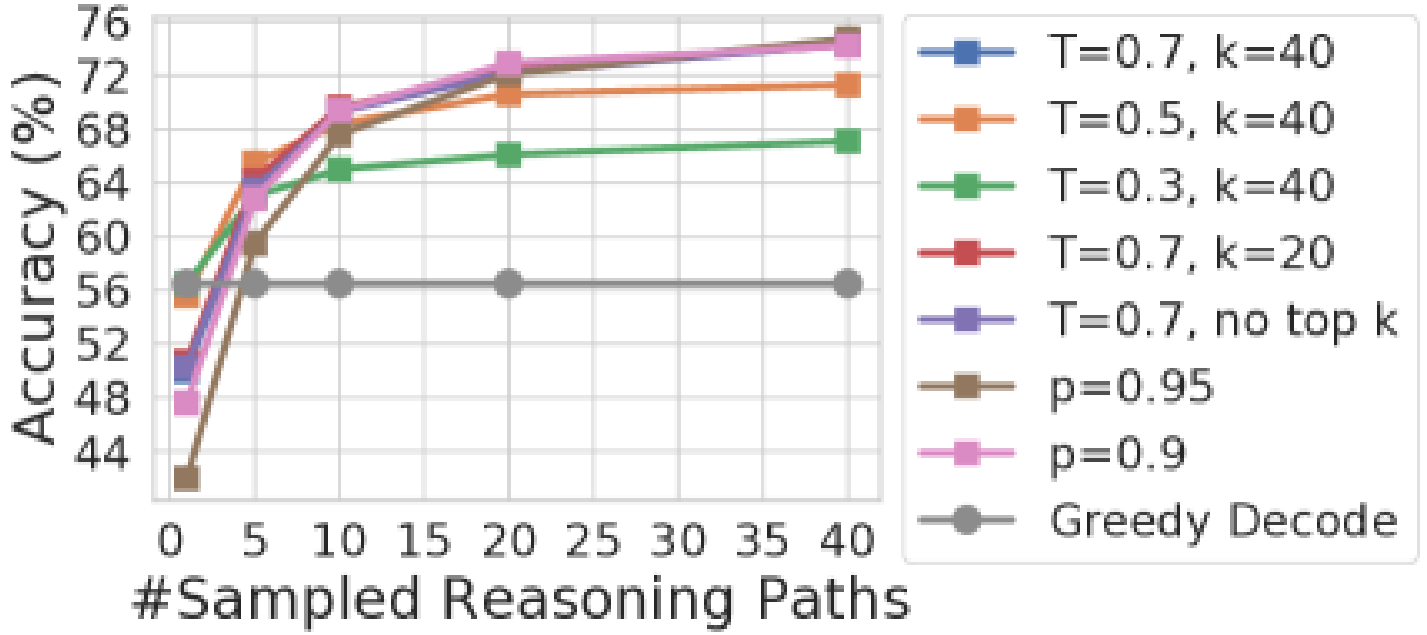
	GSM8K	MultiArith	SVAMP	ARC-e	ARC-c
CoT (Wei et al., 2022)	17.1	51.8	38.9	75.3	55.1
Ensemble (3 sets of prompts)	18.6 ± 0.5	57.1 ± 0.7	42.1 ± 0.6	76.6 ± 0.1	57.0 ± 0.2
Ensemble (40 prompt permutations)	19.2 ± 0.1	60.9 ± 0.2	42.7 ± 0.1	76.9 ± 0.1	57.0 ± 0.1
Self-Consistency (40 sampled paths)	**27.7 ± 0.2**	**75.7 ± 0.3**	**53.3 ± 0.2**	**79.3 ± 0.3**	**59.8 ± 0.2**

Self-consistency VS Sample-and-Rank

- Prompt ensemble: changing the prompt
  - A little diversity is added
  - Still essentially limited to the greedy decoding
  - Effect is limited
- Self-consistency: GOOD! GOOD! GOOD!
  - A little diversity is added
- Self-ensemble ≥ self-ensemble, change prompt, change model

Robust: Sampling Strategies and Scaling





Robust: Imperfect Prompts & zero-shot CoT

	Prompt with correct chain-of-thought	**17.1**
LaMDA-137B	Prompt with imperfect chain-of-thought	14.9
	+ Self-consistency (40 paths)	**23.4**
	Prompt with equations	5.0
	+ Self-consistency (40 paths)	**6.5**
PaLM-540B	Zero-shot CoT (Kojima et al., 2022)	43.0
	+ Self-consistency (40 paths)	**69.2**

Conclusion

- Simple and effective
  - Significantly improving the accuracy of arithmetic and common-sense reasoning
- Higher computational cost: multiple reasoning paths
  - 5–10 paths
- Use Self-consistency to generate high-quality supervised data styles
  - Fine-tune: single-shot inference can also be more accurate
- Improve the quality of reasoning chains
  - Avoid “nonsense” reasoning

Question

- How do you use your self-consistency method towards open ended questions?

- I also think they should have investigated the time-cost of self-consistency vs chain of thought.

Content

1. **Discovery:** We can unlock reasoning by prompting models to “show their work” (**Chain-of-Thought**).
2. **Robustness:** We can make this reasoning more reliable by exploring multiple reasoning paths and finding a consensus (**Self-Consistency**).
3. **Agency:** We can go a step further and have the model critique and improve its own work in a loop (**Self-Refine**).
4. **Meta-Analysis:** All these methods fall under the umbrella of “test-time compute.” How does spending more compute at inference compare to simply training a bigger model? (**Scaling LLM Test-Time Compute**).

SELF-REFINE: Iterative Refinement with Self-Feedback

Aman Madaan et al. (2023)

The Problem

LLMs often don't produce the best output on their first try.

- Generating optimal responses for complex tasks is difficult.
  - Examples: Writing engaging dialogue, optimizing code, and creative writing.
- Traditional improvement methods are expensive and complex.
  - Require large supervised training datasets.
  - Involve extra training phases or reinforcement learning (RL).
  - Need costly human annotations or reward models.

Question: Can we improve an LLM's output without extra training or data?

The Core Idea: Learn from Humans

Humans improve their work through iterative refinement.

1. **Draft:** Create an initial version (e.g., write an email, code a function).
2. **Feedback:** Review the draft and identify areas for improvement (“This sounds rude,” “This code is inefficient”).
3. **Refine:** Edit the draft based on the feedback.
4. **Repeat:** Continue the cycle until the work is satisfactory.

The main idea: Can we make an LLM follow this same process, using **itself** for feedback and refinement?

Introducing SELF-REFINE

A simple, training-free approach to improve LLM outputs.

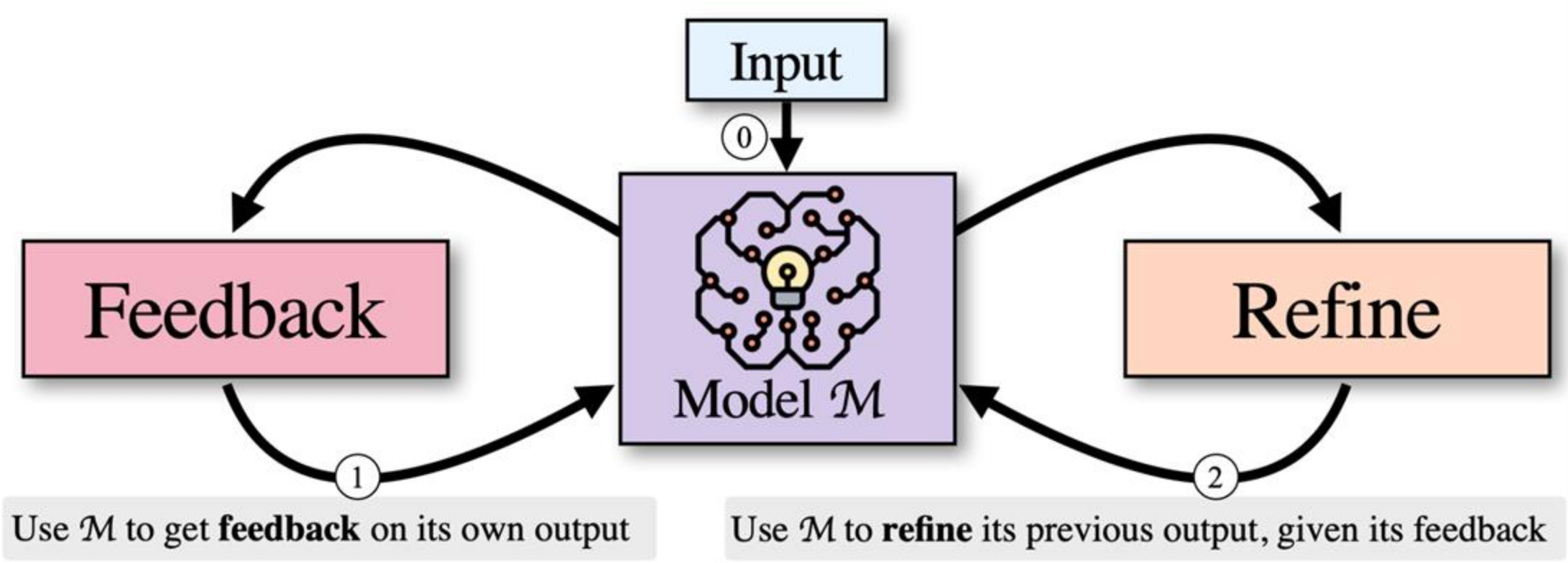
One Model, Three Roles: The same LLM is used as the

1. **Generator:** Creates the initial output.
2. **Feedback Provider:** Critiques its own output.
3. **Refiner:** Improves the output based on its own feedback.

Key Advantages:

- No supervised training data needed.
- No reinforcement learning.
- No additional models required.
- Works “out-of-the-box” with capable LLMs (like GPT-3.5 and GPT-4).

How SELF-REFINE Works



How SELF-REFINE Works

The process is a simple, iterative loop guided by few-shot prompts.

1. **Generate:** The LLM creates an initial output for a given input.
2. **Feedback:** The LLM is prompted to provide specific and actionable feedback on that output.
3. **Refine:** The LLM receives the original input, its previous output, and the new feedback, and generates an improved version.
4. **Iterate:** The Feedback → Refine loop repeats until a stopping condition is met (e.g., max iterations, or the model says “no more changes needed”).

The Algorithm in Action

Example: Code Optimization

1. Input: “Generate sum of 1, ..., N”
2. Initial Generation (y_0):

```
def sum(n):
    res = 0
    for i in range(n+1):
        res += i
    return res
```

3. Feedback (fb_0): "This code is slow as it uses brute force. A better approach is to use the formula ... (n(n+1))/2."
4. Refine (y_1):

```
def sum_faster(n):
    return (n * (n+1)) // 2
```

Evaluation Setup

Goal: Does SELF-REFINE actually improve the performance of strong base LLMs?

Models Used:

- GPT-3.5 (text-davinci-003)
- ChatGPT (gpt-3.5-turbo)
- GPT-4
- Codex (for code tasks)

Diverse Tasks (7 Total):

- Dialogue Response Generation
- Code Optimization & Readability
- Math Reasoning
- Sentiment Reversal
- Acronym Generation
- Constrained Generation

Metrics:

- Task-specific automated metrics (e.g., solve rate for math).
- Human preference (A/B testing).
- GPT-4 as a proxy for human preference.

Key Results

SELF-REFINE consistently improves performance across all tasks and models.

Average improvement is ~20% (absolute) across all tasks.

Task	Base GPT-4	GPT-4 + SELF-REFINE	Improvement
Sentiment Reversal	3.8%	36.2%	+32.4%
Dialogue Response	25.4%	74.6%	+49.2%
Code Optimization	27.3%	36.0%	+8.7%
Constrained Generation	15.0%	45.0%	+30.0%

Analysis: What Makes It Work?

1. **Feedback Quality is Crucial**
    1. Specific, actionable feedback (e.g., “Avoid repeated calculations in the for loop”) works best.
    2. Generic feedback (e.g., “Improve the efficiency”) is less helpful.
    3. No feedback (just iterating) performs the worst.
2. **Multiple Iterations Help**
    1. Performance generally increases with each FEEDBACK-REFINE cycle.
    2. Most gains are seen in the first 1-2 iterations, with diminishing returns after that.

Analysis: Key Questions

Is this just better than generating multiple samples?

- **Yes.** A single, refined output from SELF-REFINE is consistently preferred by humans over all outputs from generating multiple (k=4) initial samples.
- This shows the value is in the refinement process, not just more attempts.

Does it work with weaker models?

- **Not as well.** The approach relies on the base LLM being strong enough to both provide useful feedback and follow refinement instructions.
- Experiments with Vicuna-13B showed it struggled to follow the feedback/refine prompts consistently.

Limitations

- **Requires a Capable Base LLM:** The method’s success is dependent on the underlying model's instruction-following and reasoning abilities.
- **Closed-Source Models:** The best results are demonstrated on powerful but proprietary models like GPT-4, making reproducibility a challenge.
- **English-Only:** All experiments were conducted in English. Performance in other languages is unknown.

Conclusion & Takeaways

SELF-REFINE is a simple and powerful method to improve LLM outputs at test-time.

- It operationalizes the human creative process of iterative refinement for LLMs.
- It requires **no additional training, data, or models**, making it highly accessible.
- The results show that even state-of-the-art models like **GPT-4 are not at their performance ceiling** and can be improved with the right prompting strategy.
- This opens up possibilities for more reliable and higher-quality generation across many complex tasks.

Content

1. **Discovery:** We can unlock reasoning by prompting models to “show their work” (**Chain-of-Thought**).
2. **Robustness:** We can make this reasoning more reliable by exploring multiple reasoning paths and finding a consensus (**Self-Consistency**).
3. **Agency:** We can go a step further and have the model critique and improve its own work in a loop (**Self-Refine**).
4. **Meta-Analysis:** All these methods fall under the umbrella of “test-time compute.” How does spending more compute at inference compare to simply training a bigger model? (**Scaling LLM Test-Time Compute**).

Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters

Charlie Snell et al. (2024)

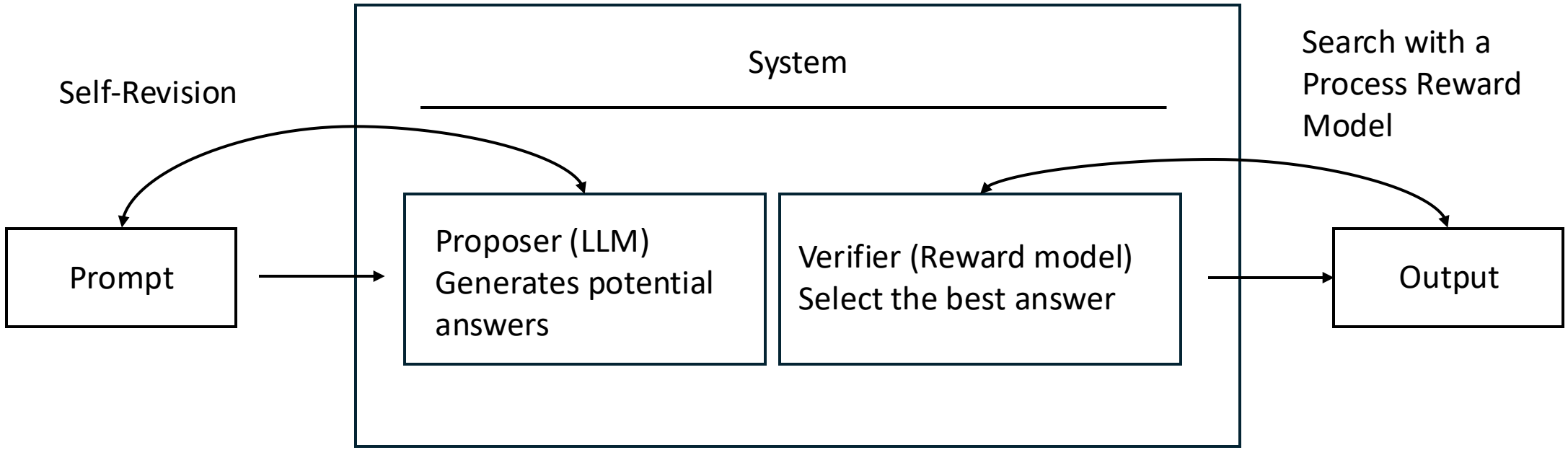
Thinking Longer or Being Smarter

- Humans think longer on hard problems. Can LLMs do the same?
- **Core Question:** If an LLM has a fixed but non-trivial amount of extra compute at inference time, how can it best use that compute to improve its answer?

Why Does This Matter?

- **Efficiency:** Can we get better performance without training even bigger models?
- **Accessibility:** Enable smaller, on-device models to achieve the performance of larger, datacenter-scale models.
- **Self-Improvement:** A path towards agents that can improve their own outputs without constant human supervision.

A Unified Framework: Proposer & Verifier

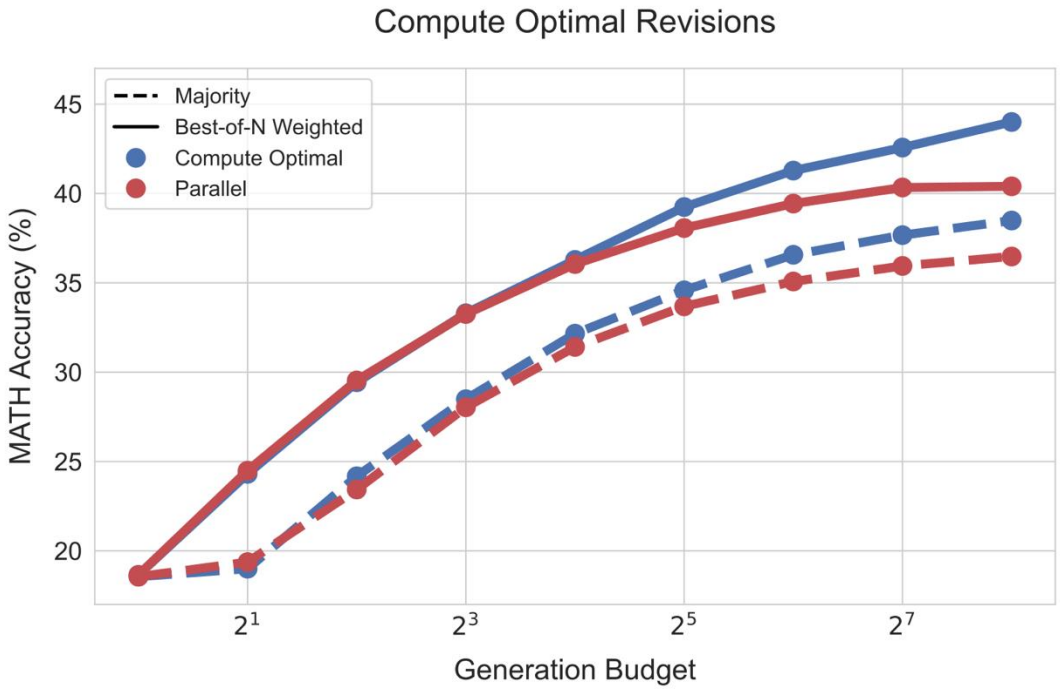


The “Compute-Optimal” Strategy

The Main Finding: The optimal way to allocate test-time compute depends critically on the prompt's difficulty.

Strategy:

- Estimate the difficulty of a prompt (e.g., using verifier scores on initial samples).
- Based on difficulty, choose the best strategy (e.g., beam search vs. Best-of-N, or sequential vs. parallel revisions).



Experimental Setup

- Datasets
  - high-school competition-level math problems
  - 12k train and 500 test questions
- Models
  - PaLM 2-S*
  - LLM
  - good at Math

Why Process Verifiers

- Which answers are correct?
- Which steps went wrong?

- Without any signal:
  - Best-of-N
  - Random sampling
- Reward Model (ORM):
  - Only focus on the final answer
  - Not resistant to deviations: result is right, but the process is all wrong
  - Tree search/beam search needs to decide the way midway
- Process Verifiers:
  - Each process success probability: Like value function in RL
  - Early Stopping, guide, and Score-weighted voting

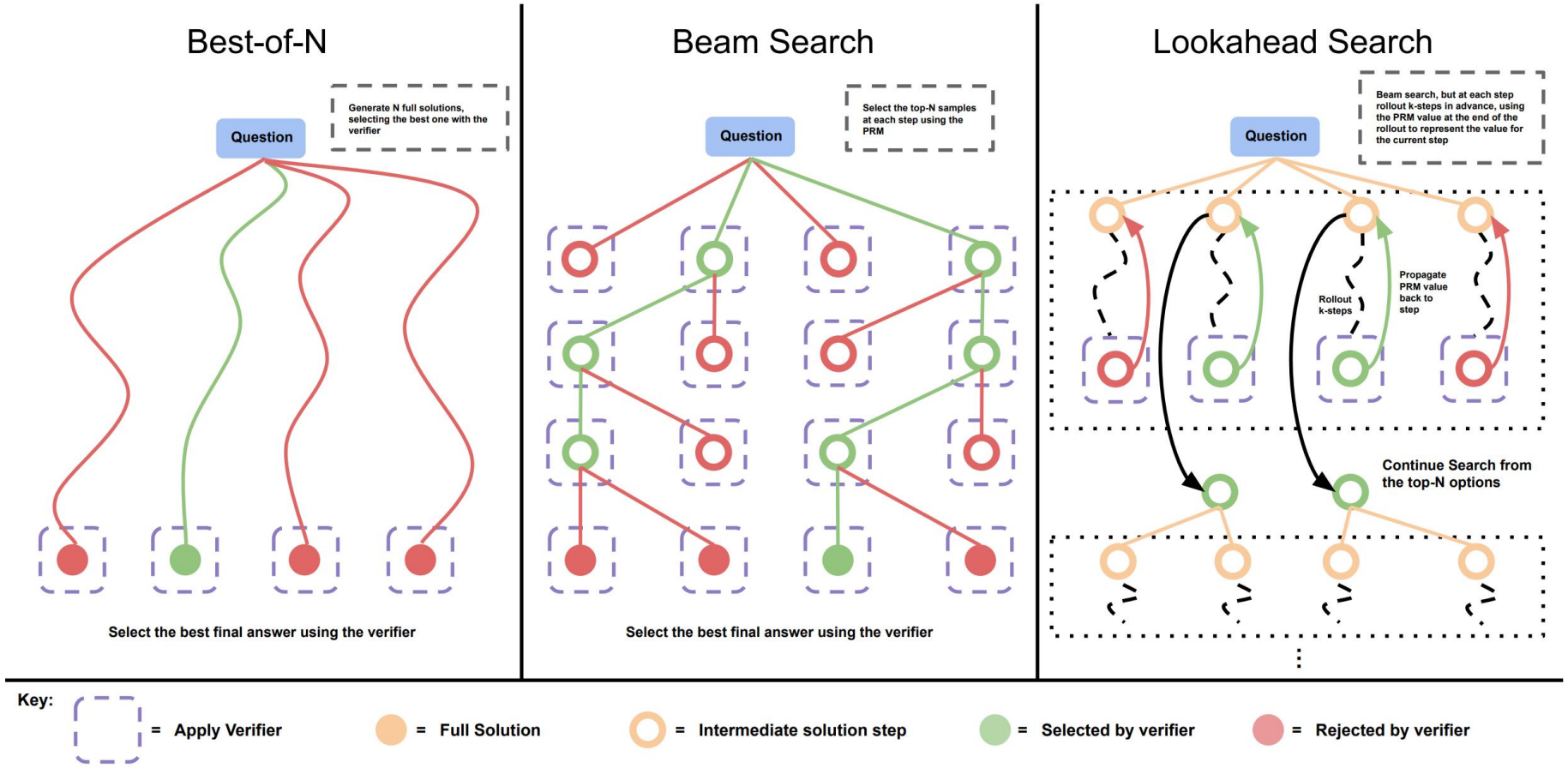
Process Verifiers (PRM) Training

- First use GPT4 PRM training data, but easy to exploit
- Do a supervised PRM
- Monte Carlo rollout

Answer aggregation

1. Aggregate each individual answer’s per-step scores
   - Use the last step as the full answer score
   - Step-wise aggregation
2. Aggregate across answers to determine the best answer
   - correctness scores of all correct answers
   - Inter-answer aggregation

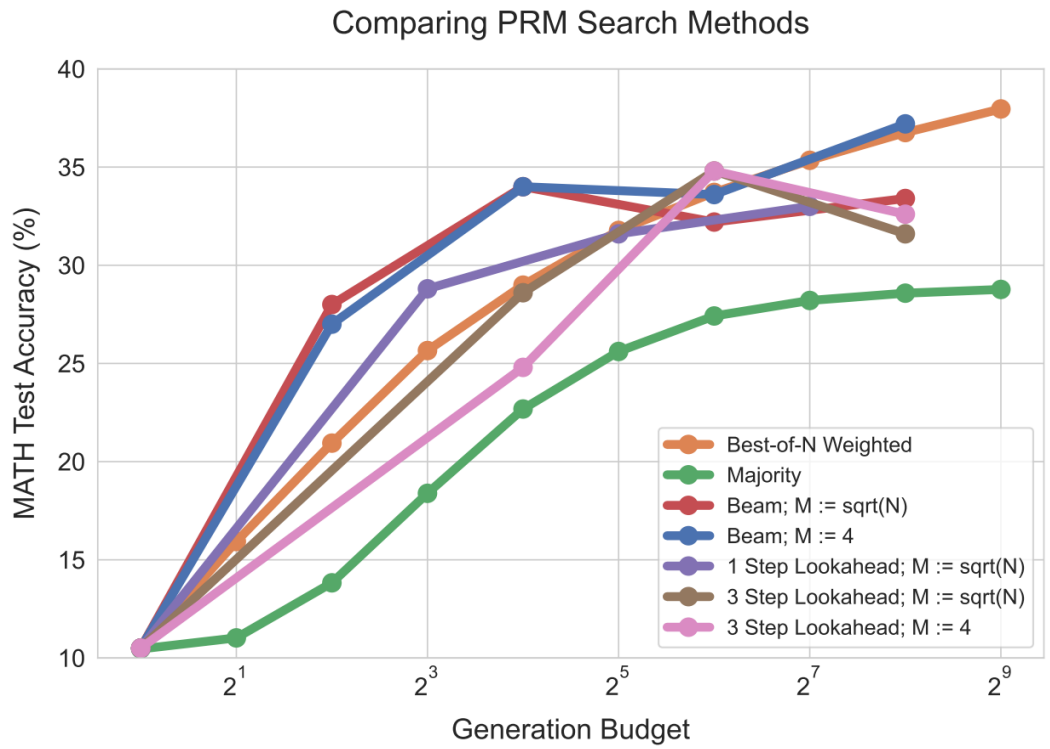
Search Methods

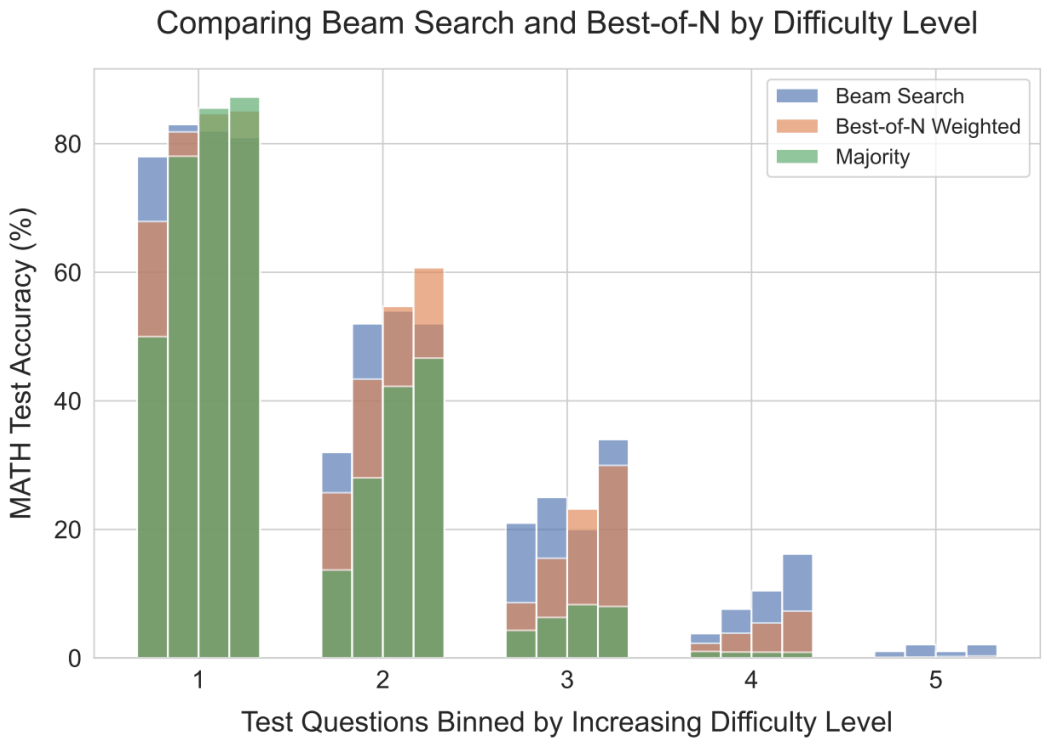


Search Methods

- Best-of-N samples N full answers and then selects the best answer according to the PRM final score.
- Beam search samples N candidates at each step, and selects the top M according to the PRM to continue the search from.
- lookahead-search extends each step in beam-search to utilize a k-step lookahead while assessing which steps to retain and continue the search from. Thus, lookahead-search needs more computation.

Search Methods Performance





Search Methods Performance

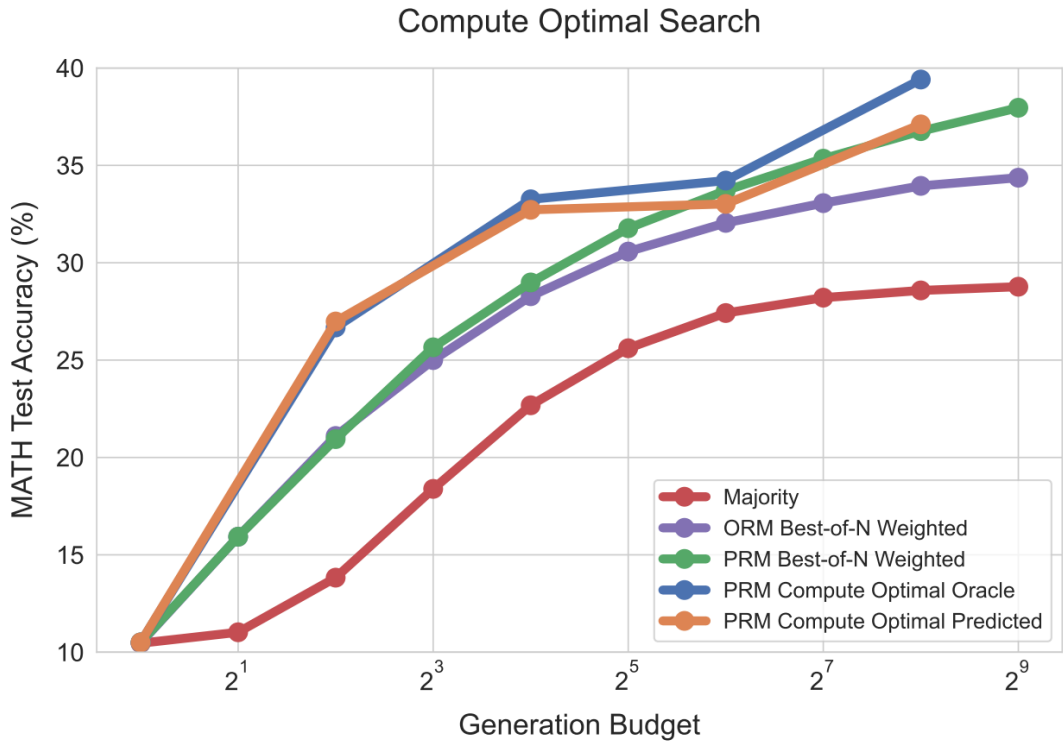
- Low generation budgets
  - Beam search performs best
- Budgets improve
  - Below the best-of-N baseline
- Easier problems (bins 1 and 2)
  - Best-of-N
  - Beam search over-optimization
- Medium difficulty problems (bins 3 and 4)
  - Beam search better

Comparing search algorithms

- Maximum budget 256
- Best-of-N
- Beam search
  - Beam width set: $\sqrt{N}$; N is the generation budget
  - Fixed beam width of 4
- Lookahead steps
  - k = 3, both settings 1) and 2)
  - k = 1, beam-search setting 1).

Comparing Compute-optimal Test-time

- Smaller budgets
  - Beam search >> Best-of-N
- Larger budgets
  - Beam search >> Best-of-N
- Lookahead search
  - The worst overall performance
- The best choice of search strategy can vary drastically as a function of this difficulty statistic.



Refining the Proposal Distribution

- **Core Idea:** Can we improve a model's answer by letting it think sequentially?
- **Traditional Method (Parallel):** "Best-of-N"– Generate N independent answers and pick the best one. This is like a broad, shallow search.
- **This Paper's Method (Sequential):** "Revisions"– Generate an answer, then generate a revision of that answer, and so on.
- **Analogy:** Instead of asking 16 different people for an answer (parallel), you ask one person to spend 16 minutes refining their single best answer (sequential).

The “Revision Model”: How It Works

Challenge: Standard LLMs are not good at self-correction on complex reasoning tasks.

Solution: Finetuning a Specialist Model

- **Data Generation:** The authors created a special dataset by pairing a correct answer with a sequence of up to four related but incorrect answers.
- **Finetuning:** They trained a PaLM 2-S* model on this data. The goal was to teach it: “Given these previous incorrect attempts, produce the correct answer.”
- **Inference:** At test time, the model can generate a chain of revisions, with each new attempt informed by the previous one. As shown in the paper, accuracy gradually improves with each revision step.

Key Finding: The Sequential vs. Parallel Tradeoff

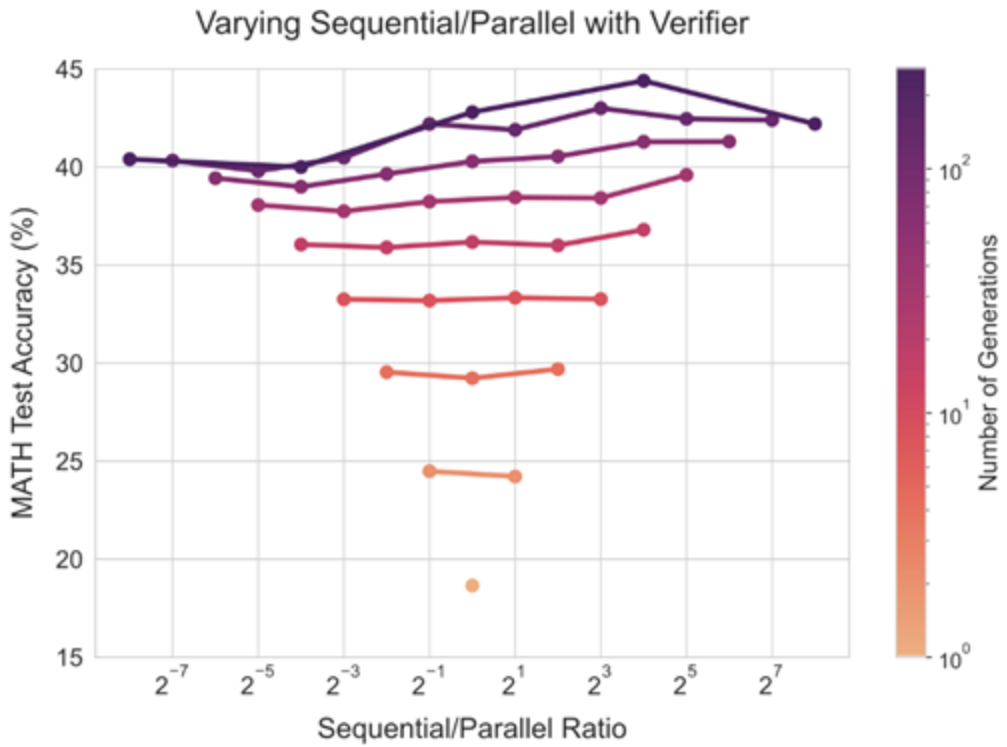
The Main Question: What's the best way to spend a fixed computing budget?

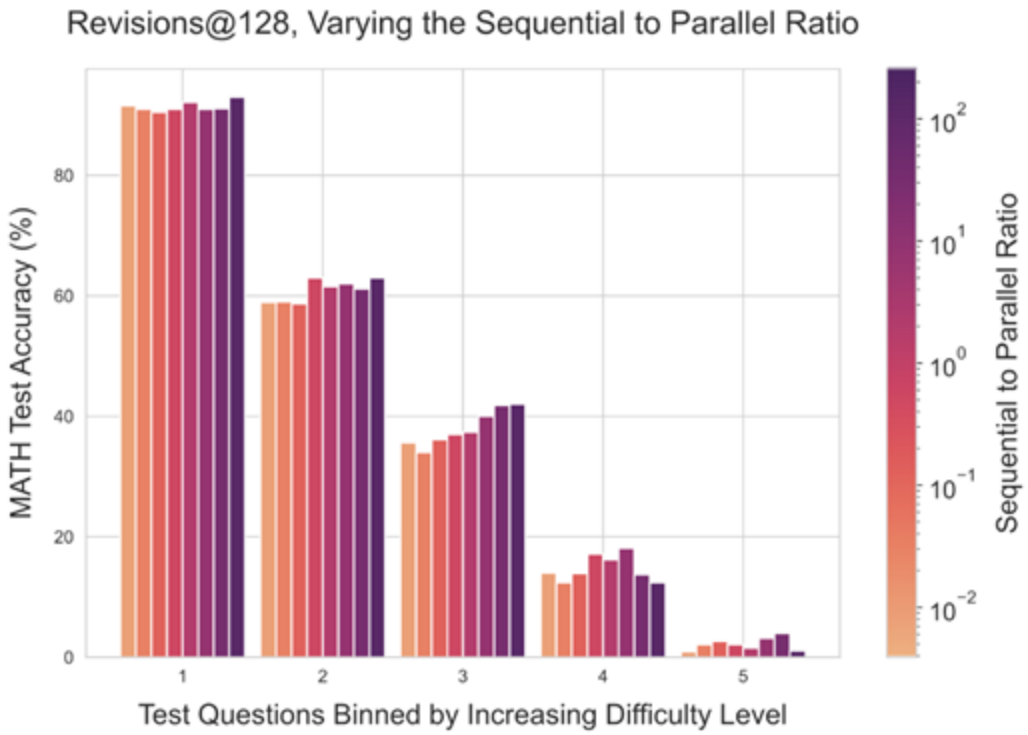
- Purely Parallel? (e.g., 16 samples)
- Purely Sequential? (e.g., 1 chain of 16 revisions)
- A mix of both? (e.g., 4 parallel chains of 4 revisions each)

The Answer: It depends on the question's difficulty!

- **Easy Questions:** Benefit most from purely sequential revisions. The model’s first guess is likely close, so refining it is the most efficient path to the correct answer.
- **Hard Questions:** Benefit most from a hybrid approach. The model needs to explore different high-level strategies (parallel) and also refine the promising ones (sequential).

Key Finding: The Sequential vs. Parallel Tradeoff





Takeaway: “Compute-Optimal Revisions”

The Strategy:

1. First, estimate the difficulty of a given question.
2. Then, allocate the compute budget to the optimal sequential/parallel ratio for that difficulty level.

The Result:

- This adaptive, “compute-optimal” strategy significantly outperforms the standard parallel best-of-N baseline.
- It can achieve the same or better accuracy while using up to 4x less test-time compute.

Conclusion: How you use your inference compute matters. Adapting the strategy to the problem's difficulty yields massive efficiency gains.

Pretraining vs. Test-Time Compute

The Ultimate Question: If you have more FLOPs (total computing power), where should you spend them?

- **Option A – Scale Pretraining:** Use the FLOPs to train a bigger, more powerful model from the start (e.g., moving from a 7B to a 100B parameter model).
- **Option B – Scale Test-Time Compute:** Use a smaller model, but give it more time and computation at inference to search, revise, and refine its answers using the optimal strategies from the previous sections.

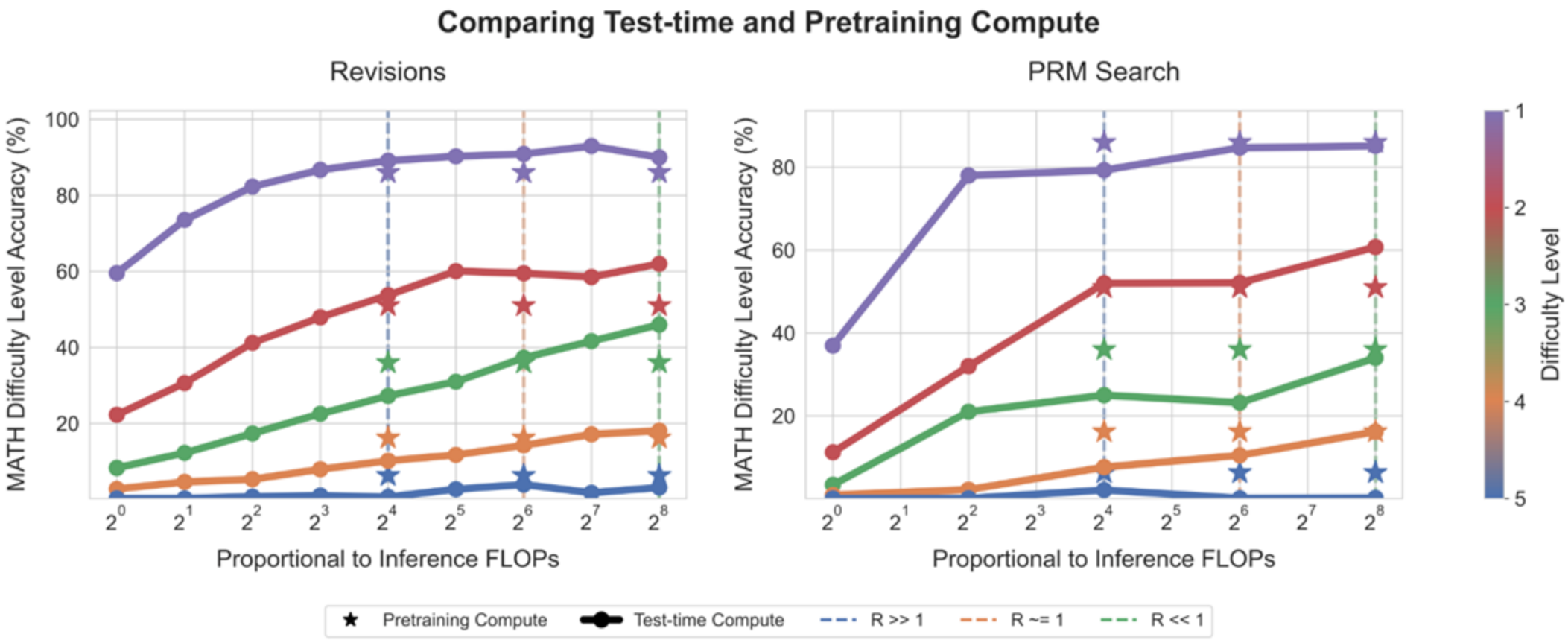
The “Exchange Rate”: When is it a Fair Trade?

The paper establishes a FLOPs-matched comparison.

It depends heavily on the ratio of **inference tokens to pretraining tokens (R)**.

- **R << 1 (Low Inference Load):** A research setting, where you generate a few high-quality samples to improve the model.
- **R >> 1 (High Inference Load):** A production setting (like a chatbot), where the model serves billions of requests.

Key Finding: There is No Single Best Answer



Key Finding: There is No Single Best Answer

The analysis shows a clear tradeoff:

Test-Time Compute is Better When:

- The questions are easy or medium difficulty. A smaller model already has the needed knowledge; it just needs time to "think."
- The inference load is low ($R << 1$).

Scaling the Pretrained Model is Better When:

- The questions are very challenging. The smaller model may lack the core knowledge to solve the problem, no matter how much time it gets.
- The inference load is high ($R >> 1$).

Conclusion & Overall Takeaways

1. **Thinking Sequentially Matters:** Iteratively revising answers is a powerful way to use test-time compute, often outperforming parallel sampling.
2. **Adaptivity is Key:** The best strategy for using test-time compute depends on the problem's difficulty. A "compute-optimal" approach can be up to 4x more efficient.
3. **It's a Tradeoff:** Test-time compute and pretraining compute are not 1-to-1 exchangeable.
4. **Future Implication:** This suggests a future where we might train smaller, more efficient models and rely on intelligent, adaptive test-time computation to achieve top-tier performance.

Q & A

Thank you for listening